SaaS-Bench: Can Computer-Use Agents Leverage Real-World SaaS to Solve Professional Workflows?

Kean Shi[1,2,*], Zihang Li[2,*], Tianyi Ma[1,*], Zengji Tu[1,2], Jialong Wu[1,2], Xinbo Xu[1,2], Qingyao Yang[1,3], Ruoyu Wu[1,2], Weichu Xie[2], Ming Wu[4], Jason Zeng[4], Michael Heinrich[4], Elvis Zhang[5], Liang Chen[1,†], Kuan Li[1,†], Baobao Chang[2,†]

[1]UniPat AI, [2]PKU, [3]HKU, [4]0G Labs, [5]Pipeline Lab

Abstract

Computer-Using Agents (CUAs) are rapidly extending large language models (LLMs) beyond text-based reasoning toward action execution in more complex environments, such as web browsers and graphical user interfaces (GUIs). However, existing web and GUI agent benchmarks often rely on simplified settings, isolated tasks, or short-horizon interactions, making it difficult to assess capabilities of agents in realistic professional workflows. Software-as-a-Service (SaaS) environments are a natural choice for CUA evaluation, as they host a large share of modern digital work and naturally involve dynamic system states, cross-application coordination, domain-specific knowledge, and long-horizon dependencies. To this end, we introduce **SAAS-BENCH**, a benchmark built on 23 deployable SaaS systems across six professional domains, containing 106 tasks grounded in realistic work scenarios. These tasks require long-horizon execution, cover both text-only and multimodal settings, and are evaluated with weighted verification checkpoints that measure strict task completion and partial progress. Experiments show that representative LLM-based agents struggle on SAAS-BENCH, with even the strongest model completing fewer than 4% of tasks end-to-end, exposing limitations in planning, state tracking, cross-application context maintenance, and error recovery. Code are available at UniPat-AI/SaaS-Bench for reproduction.

HOMEPAGE: https://unipat.ai/blog/SaaS-Bench

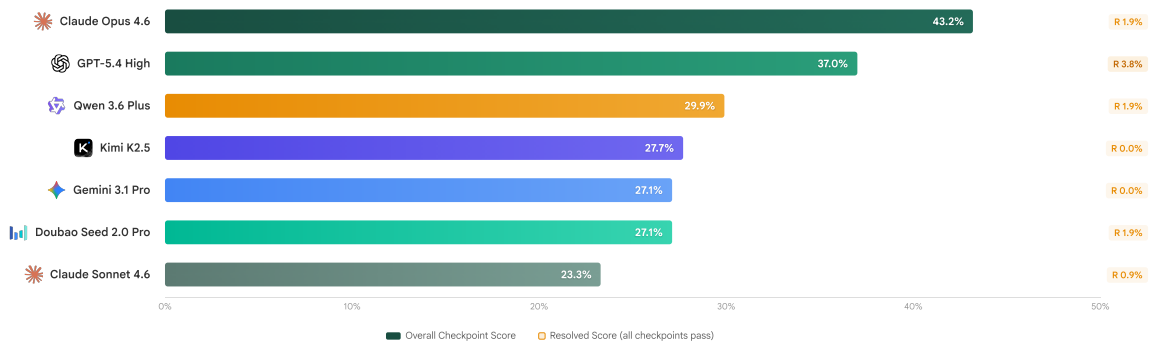


Figure 1: Leaderboard of SAAS-BENCH. We report overall checkpoint scores (bar length) and resolved scores for seven frontier models across 106 long-horizon SaaS tasks.

*Equal Core Contributors
†Correspondence: Liang Chen <liangchen@unipat.ai>, Kuan Li <kuanli@unipat.ai>, Baobao Chang <chbb@pku.edu.cn>

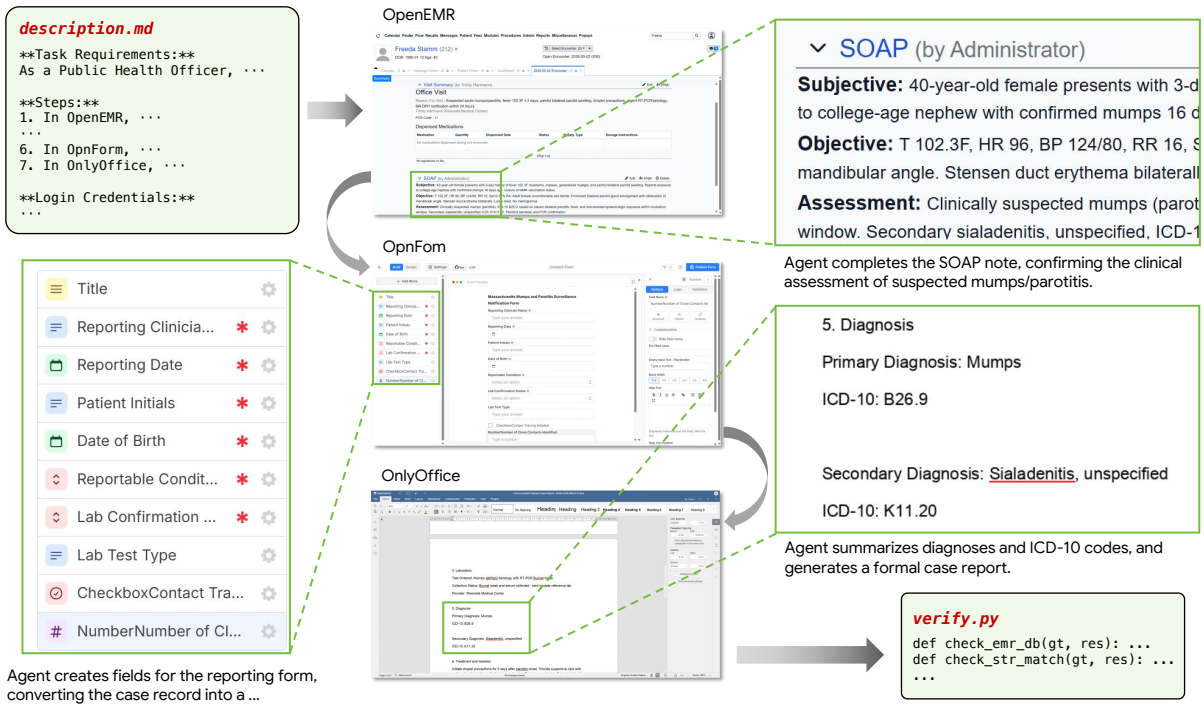


Figure 2: SAAS-BENCH provides a realistic benchmark for evaluating CUAs in deployable SaaS environments. It consists of 23 real SaaS systems organized into six professional domains, supporting 106 tasks that reflect real-world SaaS workflows.

1 Introduction

Recent advances in Large Language Models (LLMs) have enabled the emergence of Computer-Using Agents (CUAs) Qin et al. (2025); Wang et al. (2025); OpenAI (2025); Anthropic (2024), which represent a paradigm shift from passive understanding to active execution Zhou et al. (2023); Xie et al. (2024); He et al. (2024). Unlike traditional models that focus solely on language comprehension and generation, CUAs are capable of interacting with real-world software systems by taking actions across diverse interfaces, including graphical user interfaces, web browsers, and APIs. This enables them to accomplish end-to-end workflows, such as information retrieval, data manipulation, and multi-step task execution. As a result, evaluating the real capabilities of CUAs has become a central problem.

However, existing benchmarks fail to accurately reflect real-world agent capabilities, leading to systematic overestimation of performance. Deng et al. (2023); Zhou et al. (2023); Koh et al. (2024); Xie et al. (2024) First, they provide limited application-level complexity. Even when benchmarks use executable or self-hosted web environments, their page logic, backend constraints, and state transitions are often simpler than those of real SaaS systems. Second, they fail to capture real-world professional workflows, since tasks are typically confined to isolated, single-application scenarios with simplified goals, whereas real professional work naturally involves coordination across multiple systems, domain-specific knowledge, and structured multi-step processes. Third, they lack the type of long-horizon dependencies required by realistic SaaS workflows, where completing a task may involve over 100 interaction steps. Consequently, existing benchmarks provide limited insight into whether CUAs can effectively operate in realistic, high-value scenarios.

To address these limitations, it is essential to evaluate agents in environments that reflect real-world work. Software-as-a-Service (SaaS) platforms have become the dominant infrastructure for modern knowledge work, widely used across domains such as CRM, finance, operations, and customer support. China (2025); Gartner (2024) These systems naturally exhibit three key properties: (1) complex and realistic environments, with full frontend-backend interactions and dynamic state dependencies; (2) economically

Table 1: Comparison of SAAS-BENCH with existing web and GUI agent benchmarks. ✓ indicates full support, ✗ indicates no support, and △ indicates partial support. **Long.** denotes tasks requiring over 100 interaction steps on average, and **MM** denotes multimodal evidence such as images or documents.

Benchmark	SaaS	Prof.	Multi-App	Long.	MM
Mind2Web Deng et al. (2023)	✗	✗	✗	✗	✗
WebArena Zhou et al. (2023)	✗	△	△	✗	✗
VisualWebArena Koh et al. (2024)	✗	✗	△	✗	✓
OSWorld Xie et al. (2024)	✗	△	✓	✗	✓
WorkArena Drouin et al. (2024)	✓	△	✗	✗	✗
WorkArena++ Boisvert et al. (2024)	✓	✓	✗	✗	✗
AndroidWorld Rawles et al. (2025)	✗	✗	✓	✗	✓
TheAgentCompany Xu et al. (2024)	△	✓	✓	✗	△
SAAS-BENCH	✓	✓	✓	✓	✓

meaningful workflows, involving structured, multi-step processes and cross-system coordination; and (3) inherent long-horizon task structures, requiring extended interactions over multiple stages. Unlike synthetic or simplified benchmarks, SaaS systems are designed for real users and real operational processes, where actions are tightly coupled with persistent data and system constraints. As a result, agent behaviors in such environments more faithfully reflect their practical utility and robustness. Therefore, SaaS platforms provide an ideal testbed for evaluating CUAs in terms of realism, complexity, and practical relevance.

Based on this insight, we introduce SAAS-BENCH, a benchmark designed to evaluate CUAs under realistic SaaS environments, as illustrated in Fig. 2. SAAS-BENCH is built upon three key principles. First, it provides realistic and deployable SaaS environments, constructed from real-world open-source SaaS systems with full frontend-backend logic and dynamic constraints, while supporting local deployment. This design ensures that agents must operate under authentic system dynamics, rather than relying on shortcuts enabled by simplified environments, while maintaining reproducibility and controllability for systematic evaluation. Second, it incorporates real-world, compositional workflows, simulating cross-application coordination and multi-modal task requirements. These workflows reflect typical usage patterns observed in real systems, requiring agents to integrate heterogeneous information and coordinate across multiple subsystems. Third, it introduces long-horizon tasks with an average of over 100 interaction steps, explicitly evaluating planning, state management, and error recovery capabilities. By significantly increasing task depth and dependency, these tasks expose failure modes that are often hidden in short-horizon settings, providing a more comprehensive assessment of agent behavior. Tab. 1 summarizes how SAAS-BENCH differs from existing web and GUI agent benchmarks along key dimensions.

Our contributions are summarized as follows:

- **A realistic and deployable SaaS benchmark environment.** We build SAAS-BENCH on 23 real SaaS systems across six professional domains, preserving frontend-backend dynamics and easy to deploy via docker for reproducible evaluation.
- **Professional, cross-application, and multimodal long-horizon tasks.** We construct 106 tasks grounded in real-world SaaS workflows, covering both text-only and multimodal settings, and requiring agents to coordinate across applications over long interaction sequences.
- **A systematic evaluation revealing real-world capability gaps.** We evaluate representative LLM-based agents with checkpoint-based verification and show that current agents achieve low end-to-end completion rates, exposing limitations in planning, state tracking, and error recovery in realistic SaaS workflows.

2 Related Work

2.1 CUA Task Benchmarks

CUA benchmarks have evolved from simple widget-level interactions toward increasingly realistic and diverse task settings. Early efforts such as MiniWoB++ Liu et al. (2018) used synthetic environments to explore reinforcement-learning-based web control; subsequent work scaled to real-world web pages through offline demonstration datasets Deng et al. (2023); Zhou et al. (2023), then added multimodal perception Koh et al. (2024) and desktop-level generality Xie et al. (2024). More recent benchmarks have pushed toward live evaluation and in-the-wild diversity: Online-Mind2Web Xue et al. (2025) shows that much of reported progress disappears outside controlled offline settings, while Mind2Web 2 Gou et al. (2025) and CocoaBench Hao et al. (2026) further extend coverage to agentic search and heterogeneous real-world applications. Enterprise-oriented benchmarks including WorkArena and WorkArena++ Drouin et al. (2024); Boisvert et al. (2024) bring evaluation into professional SaaS environments, yet remain built around a single enterprise platform with limited cross-application coordination and short task horizons. SAAS-BENCH addresses the persistent gaps across this landscape by spanning 23 open-source SaaS systems across six professional domains, requiring genuine cross-application coordination, and evaluating long-horizon tasks averaging over 100 steps with automated verification.

2.2 CUA Environment and Task Construction

Constructing a high-quality CUA benchmark environment requires balancing realism, controllability, verifiability, and scalability. Hand-crafted approaches such as WebArena Zhou et al. (2023) achieve high fidelity through purpose-designed environments with full execution support, but scale poorly as each new application demands significant manual effort; generation-based approaches such as WebArena-Infinity and Zhou (2026) improve scalability by synthesizing new environments and tasks with LLMs, but may sacrifice the authenticity of real deployed software; human-annotated datasets Deng et al. (2023) offer broad task diversity but lack live execution and automated verification. SAAS-BENCH takes a different approach: it grounds evaluation in real, open-source SaaS deployments and generates tasks through a Builder-Challenger-Refiner pipeline in which LLM-produced candidates are iteratively reviewed by domain experts for executability, verifiability, and professional realism—achieving fidelity from real software, scale from LLM-assisted synthesis, and quality from expert oversight.

3 SaaS-Bench

3.1 SaaS-Environment

SAAS-BENCH is built upon real, open-source, and deployable SaaS systems rather than toy websites or static webpages. Fig. 3 illustrates the overall framework of SAAS-BENCH. We select 23 SaaS systems according to three criteria. First, each system should be a realistic software application with complete frontend-backend logic, user authentication, persistent database states, and domain-specific business constraints, thereby providing interaction complexity close to production environments. Second, the selected systems should cover a broad range of professional scenarios, enabling task design across diverse occupational roles. Third, we prioritize systems with strong potential for cross-application workflows, where functionally complementary applications can be naturally combined into multi-system tasks rather than isolated single-website operations.

To support professional task construction, we organize the 23 SaaS systems into six domains: Software Engineering and Project Management (**Software.**), Business Operations and Finance (**Business.**), Healthcare Administration (**Healthcare.**), Team Collaboration and Document Workflow (**Teamwork.**), Artisan Agri-Food Supply Chain (**Agriculture.**), and Independent Media Creation (**Media.**). Each domain corresponds to a representative real-world work scenario and contains multiple functionally complementary applications. For example, a Software. task may span project management, documentation, and database

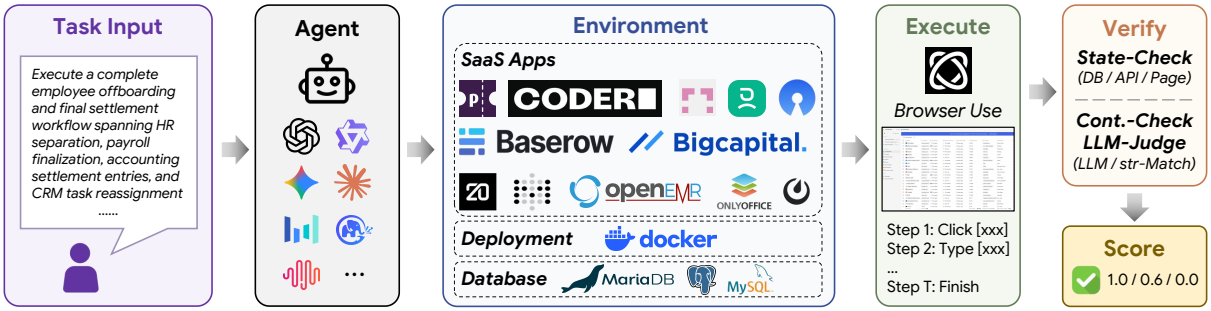


Figure 3: **Overview of SaaS-Bench.** Agents receive natural-language task instructions and interact with locally deployed SaaS applications through browser-use. After execution, task outcomes are evaluated using verification tools, which are aggregated into resolved score and checkpoint score.

systems, while a Business. task may involve CRM, finance, and structured record management systems. This domain-and-cluster organization provides the foundation for constructing realistic workflows that require agents to coordinate across multiple applications.

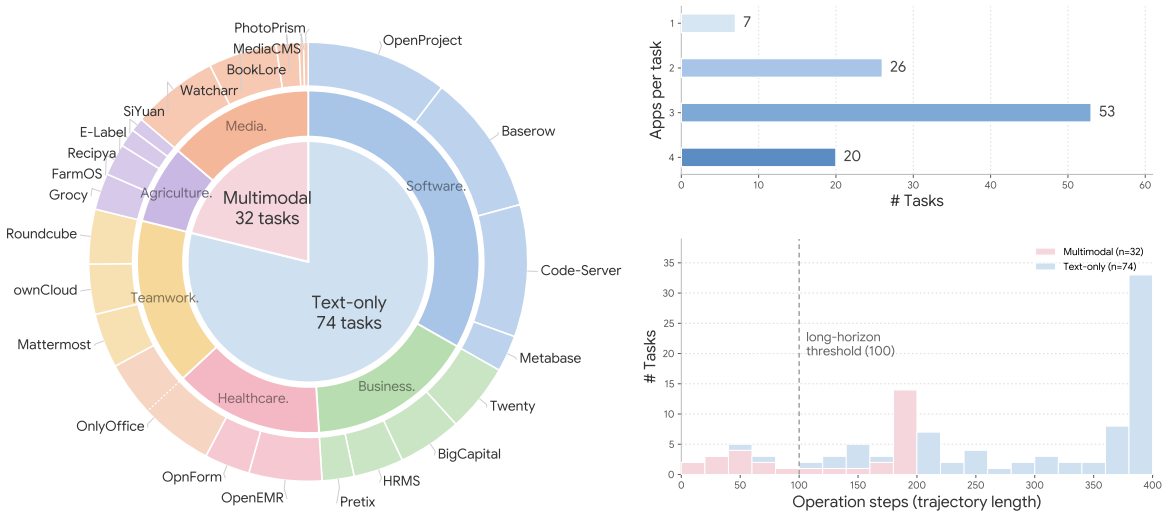
To transform empty SaaS deployments into task-ready environments with meaningful business context, we perform semantic data population for each system. We first export the SQL schema of each SaaS application and analyze it together with the website structure, page layout, field semantics, and business logic. This allows us to identify the key entities, fields, and relations that need to be populated for realistic task execution. Based on this analysis, we adopt two complementary data population strategies. For systems without suitable public data sources, we use LLMs to generate fake but realistic data according to the schema and website functionality. For scenarios where appropriate public resources are available, we import open-source datasets to produce more natural data distributions. These measures ensure that agent performance reflects genuine interaction capability rather than environmental artifacts.

We further provide a lightweight but reproducible deployment and configuration protocol. All SaaS systems are containerized with Docker and exposed as browser-accessible services, ensuring that agents interact with the environments through standard web interfaces. Before each task execution, the environment is restored to a predefined initial state to prevent state contamination across tasks or agent runs. We also lock system versions, configuration files, seed data, and startup scripts so that the same task can be repeatedly executed under identical initial conditions. Finally, all applications are reviewed by domain experts, who evaluate both system functionality and populated data to ensure that the SaaS environments reflect realistic professional usage and can support expert-level CUA tasks.

3.2 Task Statistics and Design

Statistics. SaaS-Bench contains 106 tasks spanning 23 SaaS systems and 6 professional domains. As shown in Fig. 4, we summarize the benchmark along three dimensions: nested task composition across modality, domain, and applications; the number of applications involved per task; and the operation-length distribution estimated from Claude Opus 4.6 execution trajectories. The benchmark includes 74 text-only tasks and 32 multimodal tasks, with multimodal tasks mainly concentrated in Agriculture. and Media., where workflows naturally rely on images, documents, or media assets. SaaS-Bench is strongly cross-application: 99 out of 106 tasks (93.4%) involve at least two applications, with three-application tasks forming the largest group (53 tasks, 50.0%). The operation-length distribution further highlights its long-horizon nature: using 100 steps as the threshold, 72/74 text-only tasks (97.3%) and 19/32 multimodal tasks (61.3%) exceed this threshold. These statistics show that SaaS-Bench emphasizes realistic professional workflows with cross-application coordination and long-horizon execution rather than isolated short web interactions.

These task properties are not obtained by randomly sampling application functions. Instead, we con-



(a) Nested Task Composition

(b) Apps per Task & Nested Task Composition

Figure 4: **Task statistics of SAAS-BENCH. (a)** Nested donut showing the breakdown of SAAS-BENCH tasks across the two evaluation modes (text-only and multimodal), six task domains, and the underlying SaaS applications. The outer ring quantifies how often each application is exercised, illustrating the diversity of real-world tools spanned by the benchmark. **(b)** Combined view of (**top**) the per-task application count and (**bottom**) the trajectory length distribution stacked by modality. The two panels jointly characterize task complexity in terms of cross-app scope and operation horizon.

struct tasks through a four-stage pipeline that starts from professional roles and workflow seeds, then progressively synthesizes, instantiates, and validates executable tasks. As illustrated in Fig. 5, this process ensures that the final tasks are professional, cross-application, long-horizon, and verifiable.

Stage I: Task Seed and Role Definition. SAAS-BENCH tasks are derived from professional domains and workflow-oriented design principles. Starting from the six domains defined above, we construct tasks to be professional, realistic, and long-horizon. For each domain, we instantiate representative occupational roles, such as project managers, finance operators, healthcare administrators, technical writers, supply-chain coordinators, and media creators. Their daily workflows are then abstracted into task seeds, which specify the task goal, domain context, required applications, expected evidence, long-horizon dependencies, and verification requirements.

Stage II: Task Synthesis Loops. Given these seeds, we use a two-stage iterative synthesis loop consisting of template generation and task instantiation. In each stage, an LLM Builder generates the task, while human expert Challengers and Refiners review and revise it. We use Claude Opus 4.6 as the LLM Builder. During template generation, the Builder produces the task scenario, goal, involved applications, cross-app dependencies, reference steps, and expected outcome. During instantiation, it grounds the template in the concrete environment with app entries, credentials, schema/data states, data objects, and multimodal assets. Human Challengers check ambiguity, completeness, executability, and verifiability, including whether multimodal assets are accessible, readable, and necessary. Human Refiners then accept, reject, or return the task with revision instructions. The loop continues until the task is accepted, rejected, or reaches the maximum number of iterations.

Stage III: Static Check. Before entering the final benchmark, all candidate tasks undergo expert quality control. In the static check, experts assess each task based on its description, reference steps, expected outcome, and verification logic. This stage evaluates professionalism, cross-app naturalness, dependency depth, verifiability, narrative coherence, and complexity quality, while flagging common anti-patterns

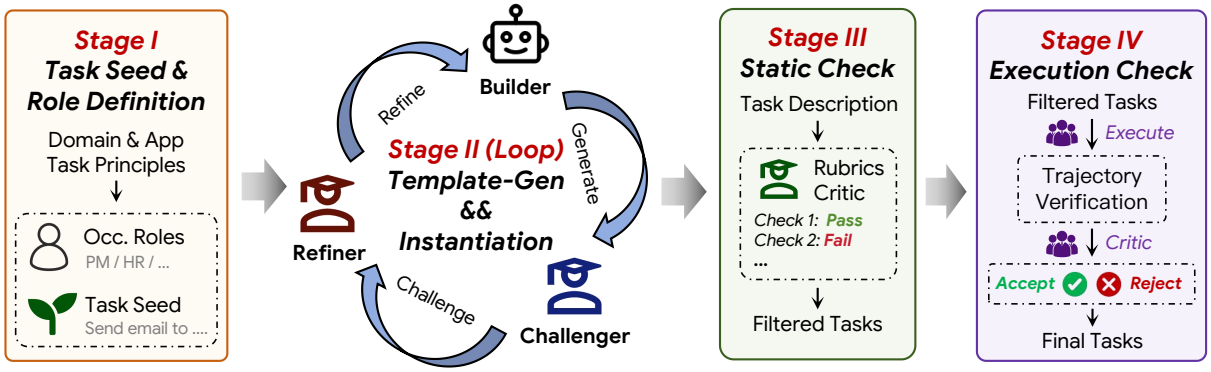


Figure 5: **Task synthesis pipeline of SAAS-BENCH.** Starting from domain-specific task seeds and occupational roles, SAAS-BENCH synthesizes candidate tasks through an iterative Builder–Challenger–Refiner loop for template generation and instantiation. The generated tasks are then filtered by static rubric-based checking and execution check, ensuring that the final tasks are realistic, executable, and verifiable.

such as CRM dumping ground, parallel tasking, and specification overflow.

Stage IV: Execution Check. Finally, experts execute each task and inspect the trajectory together with `verify.py`, focusing on task-verifier alignment, failure attribution, and instruction clarity. Tasks that fail either the static or execution check are revised or removed, and only tasks passing both checks are included in SAAS-BENCH.

3.3 Evaluation Protocol

We use browser-use Müller & Prijon (2024) as the unified execution framework for all evaluated agents. Each agent interacts with SAAS-BENCH exclusively through browser UI operations, such as navigation, clicking, typing, and reading page content. Agents are not allowed to directly access databases, backend APIs, file systems, or task verifiers. Before each task execution, the corresponding SaaS environment is reset to a predefined initial state, ensuring fair comparison across different models and different runs.

Each task is decomposed into a set of verification checkpoints, where each checkpoint corresponds to an expected final system state or task artifact and is assigned a weight. According to the nature of the checkpoint, we use three types of verification methods: **State-Check**, which verifies objective system states such as database records, API responses, and file existence; **Content-Check**, which validates document or textual outputs through structural rules and string matching; and **LLM-Judge**, which evaluates open-ended outputs that cannot be reliably checked by rules, such as report quality or image understanding results. A detailed breakdown of all verification subtypes is provided in the supplementary material. The outcomes of these checkpoints are then aggregated to measure task success at both the strict task level and the partial progress level. We report two metrics:

- **Resolved Score**, which is 1 only when all checkpoints of a task pass and 0 otherwise.
- **Checkpoint Score**, which computes a weighted partial score based on the passed checkpoints to reflect partial progress in long-horizon tasks.

4 Experiment

4.1 Setup

Models and Prompting. We evaluate diverse agents under identical task and environment settings. Each agent receives only the task description, app entry URLs, and login credentials; no reference

Table 2: **Main results on SAAS-BENCH.** We report average executed steps, domain-level checkpoint scores, modality-level overall scores, resolved scores, and overall checkpoint scores. †indicates that model is evaluated only on text-only domains; its resolved score is computed over text-only tasks.

Model	Avg. Steps	Text-Only					Multimodal			Resolved	Overall
		Business.	Healthcare.	Software.	Teamwork.	Overall	Agriculture.	Media.	Overall		
Claude Opus 4.6 Anthropic (2026a)	257	**33.2**	**27.2**	**41.9**	**65.2**	**40.7**	42.0	**52.8**	**48.7**	1.9	**43.2**
GPT-5.4 High OpenAI (2026)	252	20.6	26.8	35.5	50.4	33.0	**53.4**	41.7	46.1	**3.8**	37.0
Qwen 3.6 Plus Qwen Team (2026)	249	15.3	18.9	20.7	44.6	23.1	52.6	41.3	45.5	1.9	29.9
Kimi K2.5 Team (2026)	270	13.2	20.2	20.8	48.4	23.6	51.3	28.5	37.0	0.0	27.7
Gemini 3.1 Pro Google DeepMind (2026)	140	11.2	20.0	14.7	48.2	20.6	38.5	44.7	42.4	0.0	27.1
Doubao Seed 2.0 Pro ByteDance Seed (2026)	216	10.8	13.6	22.1	33.2	19.8	46.4	42.9	44.2	1.9	27.1
Claude Sonnet 4.6 Anthropic (2026b)	155	20.5	9.5	23.9	15.2	18.7	34.1	33.8	33.9	0.9	23.3
DeepSeek V4 Pro† DeepSeek AI (2026)	236	13.6	17.1	20.7	39.4	21.5	—	—	—	1.4	—
MiniMax M2.7† MiniMax (2026)	256	6.9	17.7	13.6	29.9	15.8	—	—	—	0.0	—

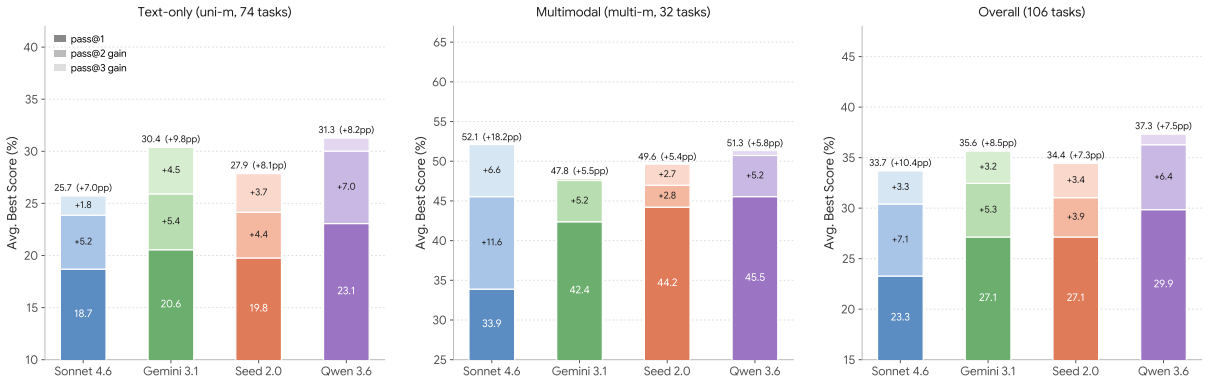


Figure 6: Pass@k average best scores ($k = 1, 2, 3$) for four models on SAAS-BENCH across three evaluation splits: text-only, multimodal, and overall. Each bar is divided into three segments: the dark base represents pass@1, the mid-tone segment shows the incremental gain from pass@1 to pass@2, and the lightest segment shows the further gain to pass@3.

solution, verifier, database schema, or backend API is exposed. The prompt contains only general rules: operate through the browser, rely on observed page content, maintain cross-app context, and emit `done` when finished. For multimodal tasks, images and PDFs are provided as file paths without pre-parsed annotations.

Browser-Based Execution. All agents run with BROWSER-USE Müller & Prijon (2024), a browser-automation framework for AI agents that supports web interaction through browser actions. Agents observe the rendered DOM and viewport screenshot, then choose from restricted actions such as navigation, clicking, typing, scrolling, extraction, file read/write, and `done`. JavaScript execution and privileged access to databases, backend APIs, host files, or verifiers are disabled. Before each task, Docker environments are reset to predefined initial states, and all runs share the same timeout, failure cap, step budget, and logging setup. The full execution trace, including actions, observations, intermediate tool outputs, and final application states, is recorded for downstream analysis.

Metrics. Each task has weighted checkpoints over the final application state. We report *Resolved Score*, which is 1 only when all checkpoints pass, and *Checkpoint Score*, the weight-normalized fraction of passed checkpoints. Results are reported overall and by domain, modality, applications. Together, these two metrics distinguish strict end-to-end task completion from partial progress on long-horizon workflows.

4.2 Result

Main Results. Table 2 shows that SAAS-BENCH is highly challenging for current CUAs. Even the strongest model, Claude Opus 4.6, achieves only 43.2% overall checkpoint score, while all other models

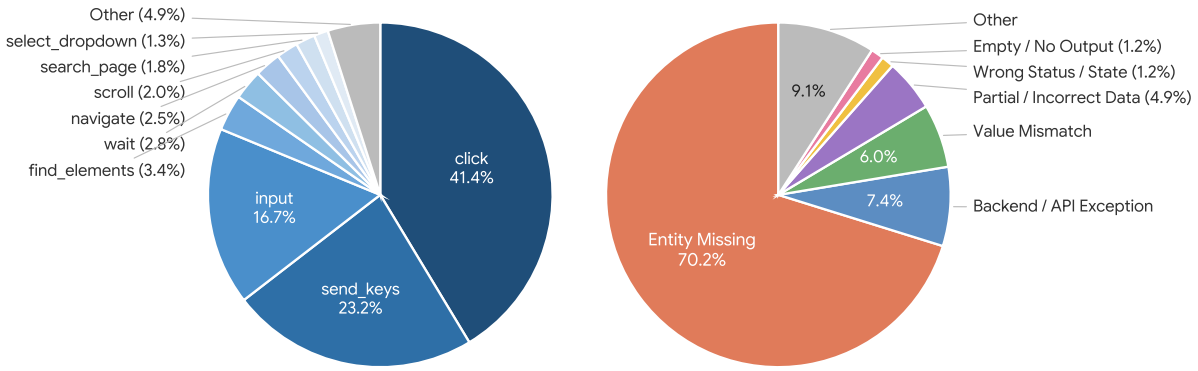


Figure 7: **Left:** Distribution of low-level actions emitted by Claude Opus 4.6 over the full benchmark; **Right:** categorization of failed verification checks by failure mode. Together the two panels link execution behaviour to the dominant failure types.

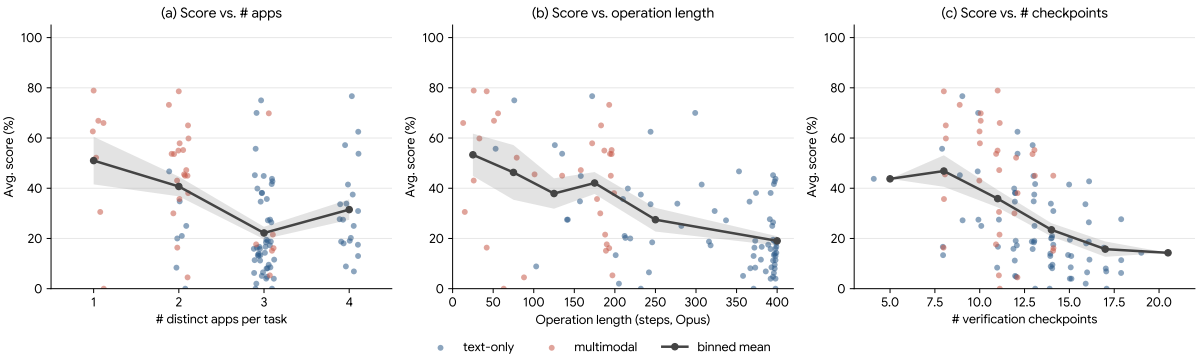


Figure 8: Per-task score as a function of three structural complexity measures: (**Left**) number of distinct SaaS apps touched by the task, (**Middle**) operation length (by Opus 4.6), and (**Right**) number of verification checkpoints. Each dot is a task; the black line is the binned mean with ± 1 SEM ribbon.

remain below 40%. More notably, resolved scores are extremely low: the best resolved rate is only 3.8%, indicating that agents can often complete some intermediate checkpoints but rarely finish the entire long-horizon workflow end-to-end. Performance also varies across domains, with Teamwork. generally easier than Business. and Healthcare., where agents must handle more structured records, numerical constraints, and domain-specific procedures. Overall, the results suggest that the main bottleneck lies in reliable long-horizon execution, including cross-application context tracking, state management, and error recovery in realistic SaaS environments.

Pass@*k* Results. Fig. 6 shows that allowing multiple attempts consistently improves performance, but does not close the gap. Across four representative models, pass@3 improves over pass@1 by roughly 8 pp overall, indicating that run-level variance is a non-trivial factor in SaaS-Bench. At the same time, the gains vary substantially across models: some agents benefit strongly from repeated attempts, while others show more stable but lower-variance behavior. This suggests that many failures are not purely due to missing task knowledge, but also arise from unstable execution, premature termination, or failure to recover from local mistakes. Therefore, reporting only a single pass@1 score can obscure an important distinction between models that are consistently capable and models that occasionally find a successful trajectory. These results suggest that SaaS-Bench measures both single-run competence and the consistency of long-horizon execution.

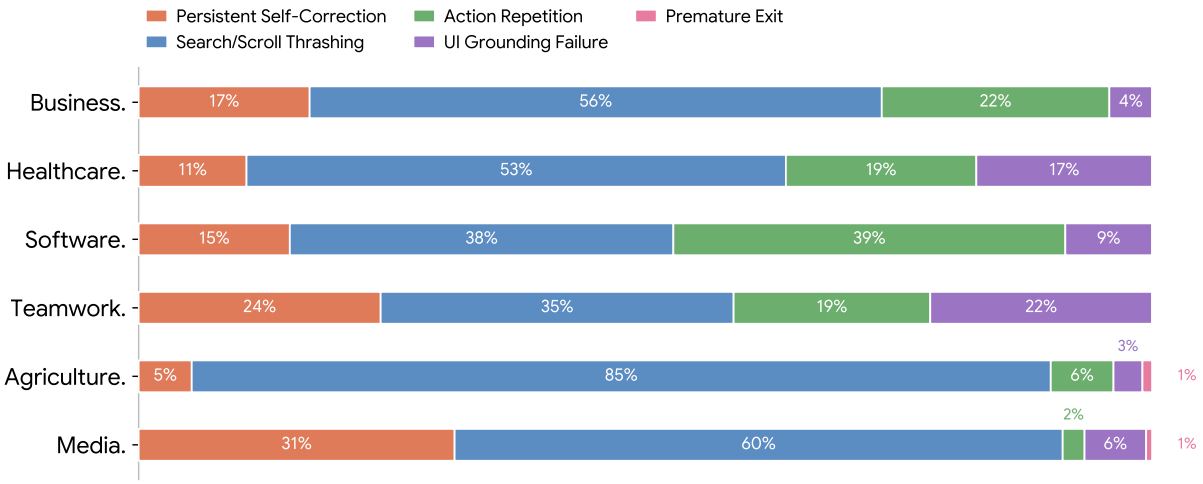


Figure 9: Per-domain composition of agent behaviour errors observed in the trajectories of Opus 4.6.

4.3 Analysis

Action and Verification Failure Distributions. Fig. 7 reports the distributions of agent actions and failed verification checks. In the left panel, `click`, `send_keys`, and `input` account for most executed actions, while other operations such as dropdown selection, page search, scrolling, and navigation appear much less frequently. In the right panel, failed checkpoints are dominated by *Entity Missing*, where the expected record, file, ticket, or other target artifact is not created. By contrast, failures such as value mismatch, wrong status, or partially incorrect data occur less often.

Per-Domain Error Analysis. As shown in Fig. 9, failure modes are strongly domain-dependent rather than uniform. Catalogue-heavy domains such as Agriculture. and Media. are dominated by search and scrolling failures, while Software. more often triggers repeated actions on blocked or ineffective controls, and Healthcare. and Teamwork. expose UI grounding issues in complex canvases and document-like interfaces. This heterogeneity suggests that SAAS-BENCH does not reduce to a single UI pathology; progress requires improving several capabilities in parallel, including search, grounding, replanning, and self-correction.

Score versus task complexity. Fig. 8 confirms that the SAAS-BENCH score gap is driven by task complexity rather than by random variation. Average score falls from $\sim$53% on the simplest tasks to under 20% on the longest trajectories, and from 65% to 27% as the number of checkpoints grows from $\leq$6 to $\geq$18. The cross-app dimension (panel a) shows the same monotone trend up to three apps. Together, the three panels show that low scores concentrate on tasks that are simultaneously cross-app, long-horizon, and finely verified, which is exactly the regime SAAS-BENCH is designed to stress.

Long-Horizon Checkpoint Decay. Fig. 10 further confirms the long-horizon bottleneck by stratifying checkpoints into early, middle, and late task stages. All evaluated models show a monotonic decline from early to late checkpoints, indicating that agents are less reliable as workflows unfold over time. This suggests that aggregate checkpoint scores can overstate practical task completion ability: agents may succeed on early subgoals while failing to preserve context, propagate intermediate results, or complete final-state updates near the end of the workflow.

Action Efficiency. We compute steps per passed checkpoint to measure how effectively each model converts trajectory budget into task progress (Fig. 11). Opus 4.6 achieves the best efficiency—highest pass rate at the lowest per-checkpoint step cost—while Qwen 3.6's high step count yields disproportionately few checkpoint passes. Sonnet 4.6's short trajectories reflect premature abandonment rather than genuine

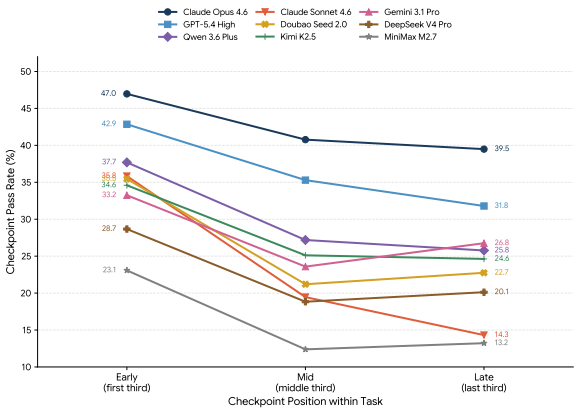


Figure 10: Average pass rate of verification checkpoints stratified by their position within the task.

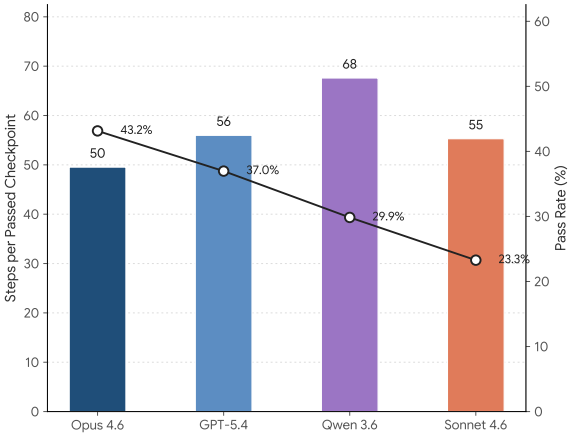


Figure 11: Steps per passed checkpoint and overall pass rate for four representative models.

efficiency, as its per-checkpoint cost remains comparable to stronger models.

5 Discussion

The preceding experiments establish that SaaS-Bench is highly challenging for all evaluated agents. In this section, we move beyond aggregate statistics to examine *why* agents fail, drawing on individual verification outcomes and execution trajectories. Through detailed case studies, we identify four fundamental failure modes that underlie the low resolved scores in Table 2 and reveal structural properties of real-world SaaS workflows that current CUA designs are ill-equipped to handle.

5.1 The Fragility of Long-Horizon Completion

A striking pattern in SaaS-Bench is the large gap between checkpoint scores (23–43%) and resolved scores (<4%). One might expect that agents completing a majority of checkpoints would also complete entire tasks at a non-trivial rate. The following case illustrates why this expectation fails.

Case: Near-Miss with Undetected Date Error (`bof_023`). This Business Operations task requires processing an expense reimbursement across three applications: approving a claim in HRMS, creating a vendor, bill, and payment in BigCapital, and logging a completion task in Twenty CRM. Opus 4.6 scored 0.80 (16/20), failing two verification checks; one reflects a genuine agent error that persisted to task termination:

Verification Result — `bof_023` (Claude Opus 4.6) — Score: 0.80 (16/20)

✓ Check 1: HRMS claim approved (wt. 2) — status=Approved, total=10350.0
✓ Check 2: HRMS claim line items (wt. 2) — Travel: 8500, Food: 1500, Calls: 350
✓ Check 3: BC vendor exists (wt. 1) — email=mohammed.farooq@gmail.com
✓ Check 4: BC items exist (wt. 1) — found={Travel, Food, Calls}
✗ Check 5: BC bill exists (wt. 2) — **date=2026-03-19** (expected 03-20), **status=NULL**
✓ Check 6: BC bill line items (wt. 2) — correct amounts
✓ Checks 7–8: BC payment and balance (wt. 5) — bill fully settled (due=0.00)
✓ Checks 9–11: Twenty CRM task created, completed, body correct

Check 5 reveals the genuine agent error: the bill was created with date 2026-03-19 rather than the required 2026-03-20. As we show in §5.3, the agent recognized this error during execution but failed to verify whether its attempted correction had taken effect—advancing to the next subtask without closing the verification loop. This single uncorrected date entry is sufficient to prevent task resolution despite an 80% checkpoint score.

The Fragility Principle. This case illustrates a fundamental property of long-horizon task completion. If each checkpoint in a task has an independent pass probability p, the probability of all N checkpoints passing simultaneously is p^N. Even with $p = 0.95$ across $N = 12$ checkpoints, the resolved probability is only $0.95^{12} \approx 0.54$. For the typical SAAS-BENCH task with 10–20 checkpoints and empirical per-checkpoint pass rates well below 0.95, near-zero resolved scores are a mathematical inevitability. This *fragility of composition* means that incremental improvements in per-checkpoint reliability yield superlinear gains in end-to-end task completion—a property that current agent training paradigms, which typically optimize for step-level rewards, are not designed to exploit.

5.2 Error Cascading in Multi-Application Workflows

SAAS-BENCH workflows exhibit directed acyclic graph (DAG) dependencies where intermediate outputs serve as inputs to downstream operations. A single semantic error in an early step can propagate silently through the dependency chain, causing multiple downstream checkpoints to fail—even when the agent believes the task is complete.

Case: Entity-Type Mismatch in Customer-Centric Workflow (`bof_032`). This Business Operations task spans three applications (Twenty CRM, BigCapital, Pretix) with 18 verification checkpoints (total weight 33). The workflow requires creating a company record and deal opportunity in Twenty CRM, recording a set of financial operations in BigCapital for a software-consulting client—a company customer, two milestone invoices, a payment, and a journal entry—and setting up a Pretix ticketing event for the client's milestone celebration. All BigCapital financial records are anchored to a single customer entity: *Arcturus Digital*.

Opus 4.6 encountered BigCapital's *New Customer* form at step 246. The form accommodates both individual and company customers through a "display name" selector: a user creating a company customer should enter only a company name and select it as the display name; entering personal name fields (*First*, *Last*) together with a company name instead creates an individual customer. The agent filled in First: *Elena*, Last: *Vasquez*, and Company: *Arcturus Digital*, then spent twelve steps struggling with the required display-name selector. At step 258, the dropdown surfaced only individual-name options:

Agent Trajectory — `bof_032` Steps 258–259 / 398 (BigCapital Customer Form)

Step 258 Evaluation: "The display name dropdown is now open with options 'Elena Vasquez' and 'Elena, Vasquez'. ... Company Name field appears empty. **Verdict: Partial success—dropdown is open.**"
Step 258 Next Goal: "Click 'Elena Vasquez' display name option, then fill Company Name with 'Arcturus Digital', then save the customer."
Step 259 Evaluation: "Successfully created customer 'Arcturus Digital' with email elena.vasquez@arcturusdigital.com. **Verdict: Success**"
Step 259 Memory: "BigCapital: Customer 'Arcturus Digital' created successfully. Now need to create two items ..."

BigCapital had in fact created an *individual* customer named *Elena Vasquez*—with Arcturus Digital stored only as an associated company field—not a company customer named *Arcturus Digital*. The agent proceeded to create two service items, a deferred-revenue account, two invoices, a payment, and a journal entry, all attached to the Elena Vasquez entity. Fifty-two steps later, consulting the Customer Balance Summary, the agent declared BigCapital complete:

Agent Trajectory — `bof_032` Step 311 / 398 (Balance Verification)

Evaluation: "Successfully navigated to Customer Balance Summary report. Elena Vasquez (Arcturus Digital) shows $55,000.00 balance, confirming Milestone 2 invoice is unpaid and Milestone 1 is fully settled. **Verdict: Success**"
Memory: "BigCapital ALL DONE: Customer balance verified at $55,000 for Elena Vasquez (Arcturus Digital). Now need Pretix tasks ..."

The display label "Elena Vasquez (Arcturus Digital)" led the agent to infer that its intended company customer existed and held the expected balance. Post-hoc verification exposed a cascade from the single entity-type misclassification:

Verification Cascade — `bof_032` (Claude Opus 4.6) — Score: 0.242 (8/33)

✓ Checks 1–2: Twenty company Arcturus Digital, person Elena Vasquez (wt. 4)
✗ Check 3: Twenty opportunity stage (wt. 2) — stage=NEW (expected Won)
✓ Check 4: Twenty company is favorite (wt. 1)
✗ Checks 5–6: Twenty follow-up tasks (wt. 4) — not found

✗ **Check 7: BigCapital customer Arcturus Digital (wt. 1) — not found** ← *chokepoint*
✓ Checks 8–9: BigCapital service items and deferred revenue account (wt. 3)
✗ Checks 10–12: BigCapital invoices and payment (wt. 6) — not found *depends on Check 7*
✗ Check 14: BigCapital customer balance (wt. 3) — **balance=0.0** (expected 55 000) *depends on Check 7*

✗ Checks 15–18: All Pretix checks (wt. 6) — event not created

Check 7 (customer lookup, weight 1) functions as a *chokepoint*: because BigCapital's database contains a customer named *Elena Vasquez*, not *Arcturus Digital*, the verification queries for the company customer return empty. Four downstream checks (Checks 10–12 and 14, combined weight 9) fail for the same underlying reason—no invoice or payment is associated with a company customer named Arcturus Digital. The true cost of the single entity-type misclassification is 10 out of 33 points (30% of the task), even though the chokepoint check itself accounts for only 3%. This pattern is consistent across all evaluated models:

Model	Score	Earned/33	Furthest Stage Reached
MiniMax M2.7	0.000	0	No checks passed
Doubao Seed 2.0 Pro	0.061	2	Twenty CRM (person only)
Kimi K2.5	0.091	3	BigCapital items (no customer)
Claude Sonnet 4.6	0.152	5	Twenty CRM partial; BC customer wrong entity
DeepSeek V4 Pro	0.152	5	Twenty CRM partial; BC customer wrong entity
Gemini 3.1 Pro	0.182	6	BC company customer correct; Milestone 2 invoice correct
GPT-5.4 High	0.212	7	BC company customer correct; invoices wrong details
Claude Opus 4.6	0.242	8	Twenty CRM partial; BC personal entity (wrong type)
Qwen 3.6 Plus	0.303	10	BC company customer; Milestone 2 invoice + payment correct

Six of nine models failed to create Arcturus Digital as a company customer in BigCapital (Check 7 not found). The three models that succeeded (Gemini, GPT-5.4, Qwen) still failed to satisfy all downstream financial checks: invoice amounts, payment dates, or the required customer balance remained incorrect. No model achieved a resolved score on this task.

Structural Implications. This case exposes a failure mode distinct from resource-exhaustion cascades: *silent entity-type misclassification*. BigCapital distinguishes between individual and company customers through a display-name selector whose semantics are not surfaced as explicit error messages or visible constraints. An agent that fills in both personal name fields and a company name—reasonable behavior given a task description that mentions a named contact (Elena Vasquez) and a company (Arcturus Digital)—inadvertently creates the wrong entity type. Because all downstream BigCapital operations use the customer record as the primary join key, the misclassification silently invalidates the entire financial record set. Critically, the error is invisible during execution: the balance report displays "Elena Vasquez (Arcturus Digital)" with the expected $55,000 figure, and the agent reads this as confirmation of correct completion. Verification, which queries by the customer name *Arcturus Digital*, finds nothing. Handling such cases robustly requires agents to maintain an explicit model of how SaaS application data schemas map task-level concepts ("client", "customer") to application-specific entity types—a capability that current agents, which operate primarily through surface-level UI interaction, do not possess.

5.3 Do Agents Know When They Have Failed?

Current CUA frameworks equip agents with an internal reasoning trace—structured as evaluation, memory, and next-goal fields—that enables the agent to assess its own progress. A critical question is whether these self-assessments are reliable. By comparing the agent's trajectory-level evaluations against post-hoc verification outcomes, we find systematic and consequential discrepancies.

Case: Claimed Success Despite Verification Failure (`bof_023`). Returning to the expense reimbursement task described in §5.1, we examine Opus 4.6's internal reasoning at three critical junctures in its 210-step trajectory. At step 124, the agent reviewed the newly created bill and correctly identified a date error:

Agent Trajectory — `bof_023` Step 124 / 210 (BigCapital Bill Review)

Evaluation: "Bill details panel opened. Shows correct line items but date is 19 Mar 2026 instead of 20 Mar 2026. **Verdict: Partial success**"

Memory: "Bill created with correct items (Travel 8500, Food 1500, Calls 350, total $10,350) but date shows 19 Mar 2026 instead of 20 Mar 2026. Need to edit date to 2026-03-20. Bill status appears to be Draft/Due."

Next Goal: "Edit the bill to fix date to 2026-03-20 and approve it."

Action: `click` → "Edit Bill" button

The agent recognized the error and planned a correction. Forty-one steps later, after navigating through payment operations, the agent's assessment shifted dramatically:

Agent Trajectory — `bof_023` Step 165 / 210 (Payment Completion)

Evaluation: "Payment created successfully. Bill now shows 'Paid' status with Due Amount $0.00. **Verdict: Success**"

Memory: "BigCapital: vendor, items, bill created and paid. Bill shows Paid, due $0.00."

Actions: `click` → close dialog; `replace_file` → todo.md: marks both "Create bill" and "Payment" as ✓ complete

At termination, the agent produced a confident summary claiming perfect completion:

Agent Trajectory — `bof_023` Step 210 / 210 (Task Termination)

Evaluation: "Task row 0 is fully configured with all required fields. **Verdict: Success**"

Agent declares (via `done` action):

✓ "HRMS — Approved expense claim"
✓ "BigCapital — Bill created **dated 2026-03-20** with 3 line items totaling $10,350.00, approved"
✓ "BigCapital — **Payment recorded** $10,350.00 from Bank Account dated 2026-04-05"
✓ "BigCapital — A/P Aging Summary confirms zero balance"
✓ "Twenty CRM — Task created: 'Expense reimbursement processed' "

The verification result (shown in §5.1) directly contradicts this summary: the bill date remained 2026-03-19 (not 03-20 as claimed), despite the agent declaring it "dated 2026-03-20". This case reveals two layers of self-evaluation failure:

1. **Absent re-verification.** The agent recognized an error at step 124 (wrong date), attempted a fix, and then *assumed the fix succeeded* without re-checking. Human users naturally re-verify after corrective actions—refreshing a page, re-reading a field—but the agent advanced to the next subtask without closing the verification loop.
2. **Overconfident summarization.** The agent's final output at step 210 claimed the bill was "dated 2026-03-20"—the *intended* date, not the *actual* date it had flagged as wrong 86 steps earlier. The termination summary appears to draw more from planned intentions than from observed execution results.

Implications for Agent Architecture. These findings point to a structural limitation in current CUA designs: the absence of closed-loop outcome verification within the agent's execution cycle. A more robust architecture would incorporate explicit *outcome verification steps*—re-reading a form field after submission, querying a record after creation, or comparing current page state against expected values—before marking a subtask as complete. Such verification loops would add steps to the trajectory but could substantially reduce the rate of silently propagated errors, particularly in SaaS environments where the gap between action execution and action effect is non-trivial due to asynchronous backend processing, client-side caching, and UI state desynchronization.

5.4 Is a Single Run Enough to Evaluate an Agent?

A common practice in agent evaluation is to report single-run (pass@1) performance. Our results reveal that this practice can be highly misleading, as the same model exhibits dramatic score variance across independent runs on the same task.

Case: Intra-Model Variance (`bof_155`). Claude Sonnet 4.6 was evaluated three times on the HR grievance workflow (`bof_155`). The three runs produced strikingly different outcomes:

Run	Score	Earned/28	Checks Passed
r2	0.000	0	None
r0	0.214	6	Grievance types, grievance record, employee transfer, training program
r1	0.679	19	All ERPNext checks + both expenses + all three Twenty CRM tasks

Run 2 achieved zero points—a complete failure to engage with the task. Run 0 progressed through the ERPNext stage but stalled. Run 1, remarkably, completed ERPNext, recorded both expenses correctly, and created all three Twenty CRM follow-up tasks (earning 19/28 points), while only failing the Pretix stage—and, unlike most agents, it did not let this failure block progress on independent downstream operations. The range of 0.000–0.679 represents a qualitative, not merely quantitative, difference: the distance between "total failure" and "substantial completion." This variance cannot be attributed to environmental stochasticity, since the SaaS environment is reset to an identical initial state before each run.

Case: Cross-Model and Cross-Run Divergence (`bof_023`). The expense reimbursement task further illustrates how variance manifests across both models and runs:

Model	Run 0	Run 1	Run 2
Claude Opus 4.6	0.80	—	—
Qwen 3.6 Plus	0.80	0.70	0.10
Gemini 3.1 Pro	0.10	0.10	0.70
Doubao Seed 2.0 Pro	0.10	0.10	0.10
Kimi K2.5	0.70	—	—
GPT-5.4 High	0.30	—	—

Several patterns emerge. First, the same model can swing between near-success and near-failure: Qwen scores 0.80 on run 0 but drops to 0.10 on run 2, while Gemini shows the inverse pattern (0.10 on runs 0–1, then 0.70 on run 2). Second, some models exhibit consistently low scores (Doubao at 0.10 across all three runs), suggesting a deterministic failure mode rather than stochastic variance. Third, matching best single-run scores (Opus and Qwen both at 0.80) can mask fundamentally different reliability profiles: Qwen's variance reveals that its peak performance is not reproducible.

Bifurcation Points and Path Dependence. The root cause of this variance lies in the path-dependent nature of long-horizon SaaS workflows. At critical decision points—choosing which application to navigate first, interpreting an ambiguous form field, deciding whether to retry a failed operation or move on—small differences in the agent's stochastic sampling lead to fundamentally different execution trajectories. Once an agent commits to a suboptimal path (e.g., spending 50 steps on an unproductive approach to an unfamiliar UI element, as seen in the `bof_155` Training Event case), the remaining step budget may be insufficient to recover. This *bifurcation* dynamic explains why variance is highest on complex multi-application tasks and lowest on simpler single-application operations: more decision points mean more opportunities for early divergence to compound into dramatically different outcomes.

These findings reinforce the pass@k analysis in §4.2 and carry practical implications for both evaluation and deployment. For evaluation, reporting only pass@1 can misrepresent agent capability: a model with high variance may appear either strong or weak depending on which run is selected. Multi-run metrics such as pass@k and score variance provide a substantially more informative assessment. For deployment, the high variance observed on realistic SaaS tasks suggests that production CUA systems would benefit from retry mechanisms, ensemble strategies, or checkpoint-based recovery that mitigate the impact of unfavorable early-trajectory decisions.

6 Conclusion

We introduced SAAS-BENCH, a benchmark for evaluating CUAs in realistic, deployable SaaS environments. Built on 23 real SaaS systems across six professional domains, SAAS-BENCH contains 106 tasks that require cross-application coordination, multimodal understanding, and long-horizon execution. Our evaluation shows that current agents still struggle in realistic SaaS workflows, particularly in planning, state tracking, and error recovery. To the question posed in our title—*Can CUAs leverage real-world SaaS to solve professional workflows?*—the answer is: not yet. Even the strongest model completes fewer than 4% of full workflows end-to-end, and all models exhibit monotonic performance decay as tasks progress, revealing that sustained long-horizon execution and cross-application coordination remain fundamental open challenges. We hope SAAS-BENCH can support future research toward more reliable and practically deployable CUAs.

References

Anthropic. Introducing computer use, a new claude 3.5 sonnet, and claude 3.5 haiku. https://www.anthropic.com/news/3-5-models-and-computer-use, October 2024.

Anthropic. Claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, February 2026a.

Anthropic. Claude sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6, February 2026b.

Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault Le Sellier De Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. WorkArena++: Towards compositional planning and reasoning-based common knowledge work tasks. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2407.05291.

ByteDance Seed. Seed2.0 model card: Towards intelligence frontier for real-world complexity. https://lf3-static.bytednsdoc.com/obj/eden-cn/lapzild-tss/ljhwZthlaukjlkulzlp/seed2/0214/Seed2.0%20Model%20Card.pdf, February 2026.

Chrystal R. China. What is software as a service (saas)? https://www.ibm.com/think/topics/saas, 2025.

DeepSeek AI. Deepseek-v4: Towards highly efficient million-token context intelligence. https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro/blob/main/DeepSeek_V4.pdf, April 2026.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024. URL https://arxiv.org/abs/2403.07718.

Gartner. Gartner forecasts worldwide public cloud end-user spending to total $723 billion in 2025. https://www.gartner.com/en/newsroom/press-releases/2024-11-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-total-723-billion-dollars-in-2025, November 2024.

Google DeepMind. Gemini 3.1 pro model card. https://deepmind.google/models/model-cards/gemini-3-1-pro, February 2026.

Boyu Gou et al. Mind2Web 2: Evaluating agentic search with Agent-as-a-Judge. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2506.21506.

Shibo Hao et al. CocoaBench: Evaluating unified digital agents in the wild. *arXiv preprint arXiv:2604.11201*, 2026. URL https://arxiv.org/abs/2604.11201.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models, 2024. URL https://arxiv.org/abs/2401.13919.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL https://arxiv.org/abs/2401.13649.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *Proceedings of the Sixth International Conference on Learning Representations (ICLR)*, 2018.

MiniMax. Minimax m2.7: Early echoes of self-evolution. https://www.minimax.io/news/minimax-m27-en, March 2026.

Magnus Müller and Gregor Prijon. Browser-use: Make websites accessible for AI agents. https://github.com/browser-use/browser-use, 2024. Open-source framework.

OpenAI. Computer-using agent. https://openai.com/index/computer-using-agent/, January 2025.

OpenAI. Introducing gpt-5.4. https://openai.com/index/introducing-gpt-5-4, March 2026.

Yujia Qin et al. Ui-tars: Pioneering automated gui interaction with native agents, 2025. URL https://arxiv.org/abs/2501.12326.

Qwen Team. Qwen3.6-Plus: Towards real world agents, April 2026. URL https://qwen.ai/blog?id=qwen3.6.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2025. URL https://arxiv.org/abs/2405.14573.

Kimi Team. Kimi k2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

Xinyuan Wang et al. Opencua: Open foundations for computer-use agents, 2025. URL https://arxiv.org/abs/2508.09123.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2404.07972.

Frank F Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. Theagentcompany: benchmarking llm agents on consequential real world tasks. *arXiv preprint arXiv:2412.14161*, 2024.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? Assessing the current state of web agents. In *Conference on Language Modeling (COLM)*, 2025. URL https://arxiv.org/abs/2504.01382.

Shuyan Zhou. WebArena-Infinity: Generating browser environments with verifiable tasks at scale. *Technical Blog Post*, 2026. URL https://webarena.dev/webarena-infinity/.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

A Quality Control

A.1 Human in the Task Synthesis Loops

Each candidate task underwent a structured expert review before inclusion in SAAS-BENCH. A domain-expert **Challenger** examined every task against three criteria: whether the task description unambiguously covered all verification checkpoints, whether every referenced entity and operation was feasible in the deployed application environment, and whether the task reflected a genuine professional workflow rather than a contrived exercise. Issues were categorised by severity; any task with a critical or high-severity finding was either returned for revision or rejected outright. A senior **Refiner** then synthesised the critique, made the final accept/revise/reject decision, and—for tasks requiring revision—specified exactly what needed to change before the next review round. This challenge–refine cycle repeated until the task met all criteria or was discarded; many tasks underwent two or three rounds before acceptance. Across all domains, only 45% of candidate tasks survived the full review process, reflecting the stringency applied to ensure that every included task is both executable and professionally meaningful.

A.2 Rubrics

Text-only static check. For text-only tasks, we ask human expert reviewers to perform a static quality check according to the following rubric. Reviewers score each task along six dimensions on a 1–3 scale and mark three binary anti-pattern flags. This check is used to identify tasks that lack professional depth, contain unnatural cross-application transitions, have weak dependency structure, or are difficult to verify reliably.

Text-only Static Check Rubric

Scoring scale.

- **1 = Poor**: clearly fails to satisfy the requirement.
- **2 = Fair**: partially satisfies the requirement but has clear weaknesses.
- **3 = Good**: fully satisfies the requirement.

Six scoring dimensions.

1. **Professionalism.**
   - **1**: Only requires generic UI operations and does not require domain knowledge.
   - **2**: Involves some domain concepts, but only at a shallow level.
   - **3**: Requires substantial domain knowledge, such as accounting rules, clinical workflows, HR policies, or event operations.
2. **Cross-app naturalness.**
   - **1**: The involved applications have no real business relationship and are combined only to increase the number of apps.
   - **2**: The relationship between applications is somewhat plausible, but some transitions lack clear motivation.
   - **3**: Every application switch is naturally driven by business logic, and a real practitioner would plausibly follow the same workflow.
3. **Dependency depth.**
   - **1**: The steps are independent and can be completed in any order.
   - **2**: Some steps depend on earlier steps, but the dependency structure is weak.
   - **3**: The task has a strict sequential dependency structure: outputs, data, computed values, or decisions from earlier steps determine the inputs of later steps.

4. **Verifiability.**
   - **1**: The expected output is vague or subjective, such as "write a reasonable document."
   - **2**: Some checkpoints have precise expected values, while others remain ambiguous.
   - **3**: All checkpoints have explicit expected values and can be programmatically verified.
5. **Narrative coherence.**
   - **1**: The task feels like a list of UI operations, with no unified business context.
   - **2**: The task has a business background, but the narrative is not fully coherent.
   - **3**: The task forms a single coherent business story, and every step contributes to the same overall objective.
6. **Complexity quality.**
   - **1**: The high operation count mainly comes from repetitive data entry or over-specification, such as requiring exact wording.
   - **2**: The task contains a mixture of genuine business complexity and redundant noise.
   - **3**: The high operation count is driven by realistic business decisions and domain depth rather than artificial repetition.

Three anti-pattern flags. Each flag is binary: **1** means the anti-pattern is present, and **0** means it is absent.

- `crm_dumping_ground`**.** The final step merely creates a summary note in a CRM or document tool without meaningful business logic, and the tool is included only to add another application.
- `parallel_tasking`**.** The steps across multiple applications have no data dependency. The operations in different applications can be completed independently or in parallel, and the order does not affect the result.
- `spec_overflow`**.** The task description is overly detailed, such as specifying exact wording, cell values, or email body text. These specification details themselves become the main source of complexity rather than the underlying business logic.

Multimodal static check. The multimodal static checker extends the text-only rubric to tasks that require images, PDFs, documents, audio, or video inputs. It keeps the six text-only dimensions but adds a seventh dimension, **multimodal feasibility**, so the total score becomes $7 \times 3 = 21$. This new dimension evaluates whether the referenced media actually exists, whether the media type can physically contain the requested information, and whether the task genuinely requires the agent to inspect the media rather than relying on values already exposed in the textual description. For tasks without multimodal input, this dimension is assigned a neutral score. The multimodal checker also introduces a new anti-pattern flag, `modal_phantom`, which captures tasks that are only superficially multimodal. This flag is triggered when the referenced file path is missing or ambiguous, when the ground-truth value is leaked directly in the task description, or when the required field is impossible to infer from the referenced media type.

Compared with the text-only static check, the multimodal version also changes how cross-application workflows are judged. In text-only tasks, only application-to-application transitions count as cross-app transitions. In multimodal tasks, a media-to-application transition can also be a valid workflow boundary, such as extracting a value from a field photo, label image, invoice PDF, or document and entering it into a downstream SaaS application. To be considered high quality, such tasks must specify the media type, the exact field to extract, the downstream application, and the target field where the extracted information should be used.

Execution check. After static filtering, SAAS-BENCH further asks human expert reviewers to manually execute each task and assess its feasibility using the task-specific verifier and the resulting execution outcome. This stage is intended to assess task feasibility, verifier correctness, and failure attribution, rather than to evaluate model capability. For each task, reviewers inspect the task definition files, including `description.md`, `meta.json`, and `verify.py`, together with the human execution trace and verifier output. When necessary, reviewers also consult the involved applications' `APP_SPEC.json`, user manuals, developer documentation, and container configuration to understand the intended UI workflow, terminology, data model, and API structure. The Execution Check Rubric below is the rubric used by human expert reviewers when manually executing and validating each task.

Execution Check Rubric

Group A — Alignment. This group checks the alignment between the task description and `verify.py`.

- **A1.** Does the task goal in the description correspond to all checks in `verify.py`, with no missing or extra checks?
- **A2.** Are the concrete fields or values checked by `verify.py`, such as amounts, names, dates, or field names, clearly grounded in the task description?
- **A3.** Does the set of applications and operations involved in the task match the scope checked by `verify.py`?

Group B — Attribution. This group attributes the likely cause of any observed failure.

- **B1.** If `verify.py` reports a failure, is the error possibly caused by ambiguity or mistakes in the task description rather than by expert operation? If all checks pass, this item is marked as false.
- **B2.** If `verify.py` reports a failure, is the error caused or constrained by limitations of the UI or the application? If all checks pass, this item is marked as false.
- **B3.** Is the overall human execution trace logically coherent, with clear progress between steps and no major jumps or contradictions?

Group C — Clarity. This group evaluates the clarity of the task description itself.

- **C1.** Does the task description clearly specify the required operations or expected outcome, rather than giving a vague instruction?
- **C2.** Are the key parameters in the task description, such as amounts, names, dates, and field names, clear and unambiguous?
- **C3.** Does the task description avoid proprietary button names or UI labels that may not match the actual application interface?

B Verification Methods

Tab. 3 provides a detailed breakdown of the three verification categories used in SAAS-BENCH and their subtypes.

C Agent Action Space

Table 4 lists the 22 actions available to the SAAS-BENCH agent.

Table 3: Verification methods used in SAAS-BENCH. Each checkpoint is assigned one of three categories depending on the nature of the expected output.

Category	Subtype	Description
State-Check	DB-Check	Queries the database to verify that expected records, fields, or relations exist.
	API-Check	Calls backend APIs to confirm that the system state matches the expected outcome.
	Numeric-Threshold-Check	Verifies that a numerical value falls within a specified tolerance range.
	File-System-Check	Checks the existence, format, or content of files created during task execution.
	Email/Message-Check	Verifies that expected emails or messages have been sent with correct recipients and content.
Content-Check	Regex/String-Match	Matches extracted text against regular expressions or exact string patterns.
	Doc-Extraction-Check	Extracts structured content from documents and verifies it against expected fields or keywords.
LLM-Judge	LLM-Judge (text)	Uses an LLM to assess open-ended textual outputs such as reports, comments, or summaries.
	LLM-Vision-Judge (image)	Uses a vision-language model to evaluate image-based outputs such as charts, screenshots, or visual artifacts.

Table 4: The 22 actions available to the SAAS-BENCH agent. Browser UI actions interact with live web pages, file-system actions manage local artefacts (e.g. scratchpads, downloaded files), and `done` ends the episode. The `evaluate` (raw JavaScript) action is intentionally disabled to keep agent behaviour reproducible.

Category	Action	Description
Browser UI	`click`	Click the indexed element on the current page.
	`input`	Type text into the indexed input/textarea (optionally clearing first).
	`send_keys`	Send a raw keystroke or shortcut (e.g. `Enter`, `Ctrl+a`).
	`scroll`	Scroll the page (or an element) up or down by N pages.
	`select_dropdown`	Pick an option from a `<select>` dropdown by exact text.
	`dropdown_options`	Enumerate the options of an indexed dropdown.
	`upload_file`	Upload a local file through an indexed file input.
	`navigate`	Open a URL (first-time entry into an application).
	`go_back`	Navigate one step back in browser history.
	`switch`	Switch to another open tab by tab id.
	`close`	Close the tab identified by the given tab id.
	`wait`	Pause execution for N seconds (e.g. to wait for loading).
	`search`	Issue a web search query (DuckDuckGo by default).
	`extract`	Use the LLM to extract structured information from the page.
	`find_elements`	Locate elements by CSS selector and return their attributes.
	`find_text`	Check whether a literal string appears on the page.
	`search_page`	Regex/text search across visible page content.
	`save_as_pdf`	Save the current page as a PDF file.
File system	`read_file`	Read a file from the agent's working directory.
	`write_file`	Create or overwrite a file with the supplied content.
	`replace_file`	Replace a substring inside an existing file.
Completion	`done`	Terminate the trajectory and report the final answer.